```cpp
//-----------------------------------------------------------------------------------------------------------------
// ASSOCIATIVE CONTAINERS
//-----------------------------------------------------------------------------------------------------------------
#include <iostream>
#include <map>
#include <unordered_map>
#include <utility>

using std::map;
using std::unordered_map;
using std::cout;
using std::endl;
using std::pair;
using std::string;
void print(const map<int, string> & m);
void print(const unordered_map<int, string> & um);

int main(){

    // Student ID and name
    map<int, string> m;
    unordered_map<int, string>  um;

    // all containers are empty
    // Let's store
    //  1, "Elem 1"
    //  2, "Elem 2"
    //  4, "Elem 4"
    m[1] = "Elem 1";
    m[2] = "Elem 2";
    m[4] = "Elem 4";

    // Let's copy m to um
    um.insert(m.cbegin(),m.cend());

    // writing  elements to cout (read only)
    cout << "Print m" << endl;
    print(m);
    cout << "Print um" << endl;
    print(um);

    // let's try to insert again 4
    cout << "Trying to insert 4 again" << endl;
    m.insert(std::make_pair(4,"New val for 4"));
    um.insert(std::make_pair(4,"New val for 4"));

    cout << "Print m" << endl;
    print(m);
    cout << "Print um" << endl;
    print(um);

    // Let's change elem 4!
    cout << "Changing 4" << endl;
    m[4] = "New val for 4";
    um[4] = "New val for 4";

    cout << "Print m" << endl;
    print(m);
    cout << "Print um" << endl;
    print(um);

    // Let's look for 1 and print
    if (m.find(1)!=m.end())     {
        cout << "Yeah!  1 is here and it's element is: " << m[1] << endl;
    }
    if (um.find(1)!=um.end())   {
        cout << "Yeah!  1 is here and it's element is: "<< um[1] << endl;
    }

    // Let's look for 5 and print
    if (m.find(5) == m.end())   {
        cout << "Unfortunately 5 is not here "<< endl;
    }
    if (um.find(5)==um.end())   {
        cout << "Unfortunately 5 is not here "<< endl;
    }

    // Try to access elem 5!  We are inserting empty string
    cout << m[5] << endl;
    cout << um[5] << endl;
    cout << "Print m" << endl;
    print(m);
    cout << "Print um" << endl;
    print(um);

    /*
    // Try to access elem 6!  This will raise an exception
    cout << m.at(6) << endl;
    cout << um.at(6) << endl;
    */
```

```
    // delete all elements
    m.clear();

    // All um are deleted through destructors ;)!!!
    return 0;
}

// if you want to change elements rely on iterator instead of const_iterator, but same loops!
void print(const map<int, string> & m){
    for(map<int, string>::const_iterator it = m.cbegin(); it != m.cend(); it++)
        cout << it->first << " " << it->second  << endl;
}

// same for unoredered_map and not very different from, e.g., vectors!
void print(const unordered_map<int, string> & um){
    for(unordered_map<int, string>::const_iterator it = um.cbegin(); it != um.cend(); it++)
        cout << it->first << " " << it->second  << endl;
}
```